Örebro universitet
Akademin för naturvetenskap och teknik
Thomas Padron-McCarthy (thomas.padron-mccarthy@oru.se)

Tentamen i

Kompilatorer och interpretatorer

för Dataingenjörsprogrammet m fl

måndag 31 oktober 2011

Gäller som tentamen för:
DT3004 Datateknik C, Kompilatorer och interpretatorer, provkod 0100

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 30. För godkänt betyg (3 respektive G) krävs 15 poäng.
Resultat:	Meddelas på kursens hemsida eller via e-post senast måndag 21 november 2011.
Återlämning av tentor:	Efter att resultatet meddelats kan tentorna hämtas på universitetets centrala tentamensutlämning.
Examinator och jourhavande:	Thomas Padron-McCarthy, telefon 070 - 73 47 013.

- Skriv tydligt och klart. Lösningar som inte går att läsa kan naturligtvis inte ge några poäng. Oklara och tvetydiga formuleringar kommer att misstolkas.
- Skriv den personliga tentamenskoden på varje inlämnat blad. Skriv *inte* namn eller personnummer på bladen.
- Skriv bara på en sida av papperet. Använd inte röd skrift.
- Antaganden utöver de som står i uppgifterna måste anges.
- Skriv gärna förklaringar om hur du tänkt. Även ett svar som är fel kan ge poäng, om det finns med en förklaring som visar att huvudtankarna var rätt.

LYCKA TILL!

Uppgift 1: Grunder om kompilatorer (6 p)

När man skriver ett någorlunda stort program är det bra att dela upp det i moduler. Det gäller även kompilatorer. Vi gör ett försök att dela upp kompilatorn i moduler, och åstadkommer den här listan:

1. debugger
2. editor
3. kodgenerering
4. lexikalisk analys ("scanner")
5. länkning
6. maskinberoende optimering
7. maskinoberoende optimering
8. mellankodsgenerering
9. semantisk analys
10. symboltabellen
11. syntaktisk analys ("parser")

a) (1 p) Det blev fel i listan ovan. Vilka av de uppräknade modulerna ingår *inte* i kompilatorn?

a) (1p) En del av de uppräknade modulerna brukar kallas "faser". Vad menas med en fas?

a) (0.5 p) En av modulerna ingår i kompilatorn, men är inte en fas. Vilken?

c) (0.5p) I listan ovan är modulerna ordnade i bokstavsordning. Lista *faserna* i rätt ordning.

d) (3p) Indata till den första fasen består av källprogrammet i textform (alltså en sekvens av tecken), medan utdata från den sista fasen utgörs av ett målprogram uttryckt i maskinspråk eller assemblerspråk. Beskriv alla fasernas indata och utdata.

Uppgift 2: Mellankod (6 p)

Det här är ett programavsnitt i ett C-liknande språk:

```
if (a < b) {
    c = d / e / f;
}
else {
    while (g > h) {
        i = j;
        k = l / (m / n);
    }
    o = 17;
}
```

Översätt ovanstående programavsnitt till *två* av följande tre typer av mellankod.

a) ett abstrakt syntaxträd (genom att rita upp trädet!)

b) postfixkod för en stackmaskin

c) treadresskod

Observera: Det finns tre deluppgifter i uppgiften ovan. Välj ut och besvara (högst) *två* av dessa. (Skulle du svara på alla tre, räknas den med högst poäng bort.)

Uppgift 3: En oändlig loop (2p)

while-loopen i uppgiften ovan ser ut att ha ett problem. Om villkoret **g > h** är sant (och **i** eller **k** inte på något oväntat sätt refererar till **g** eller **h**), kommer loopen aldrig att avslutas.

a) Vad innebär detta för mellankodsgenereringen i uppgiften ovan? Förklara också varför!

b) Vad innebär detta för kompileringen i övrigt? Dvs, beskriv om andra delar av kompilatorn på något sätt hanterar att loopen är oändlig.

Scenario till resten av uppgifterna

Vi åker runt och avlyssnar radiomeddelanden:

Vi konstruerar ett enkelt språk för att ange var och när vi avlyssnat. Man använder nyckelordet **avlyssnat** för att ange en avlyssning, med tid och plats. Platser kan anges med koordinater, men man kan också namnge platser med nyckelordet **plats**. Man använder nyckelordet **slut** för att ange slutet på hela inmatningen.

Här är ett komplett exempel:

```
avlyssnat 9:10:22.103 59.274120 15.2066
plats Örebro = 59.274120 15.2066
avlyssnat 15:07:28.210 Örebro
plats Örebro-universitet = 59.254320 15.24625
plats Brickeberg = 59.2465 15.2415
plats Tripoli = 32.902222 13.185833
avlyssnat 16:01:01 Tripoli
slut
```

Kommandona ska kunna skrivas på fritt format, vilket betyder att blanktecken och radslut inte spelar någon roll, annat än för att skilja orden från varandra.

Vi antar att platsnamn alltid består av ett enda ord.

Uppgift 4: Grammatiker (4 p)

a) (1p) Vilka tokentyper ingår i inmatningsspråket från uppgiften ovan?

b) (3p) Skriv en grammatik för språket. Använd terminalerna från deluppgift a. Om grammatiken inte lämpar sig för implementation i form av en prediktiv recursive-descent-parser, ska du också transformera den så den passar.

Uppgift 5: Parsning (6 p)

Skriv en prediktiv recursive-descent-parser för inmatningsspråket ovan. Gör detta i ett språk som åtminstone liknar något vanligt känt programmeringsspråk, till exempel C. Du behöver inte skriva exakt korrekt programkod, men det ska framgå vilka procedurer som finns, hur de anropar varandra, och vilka jämförelser med tokentyper som görs. Förklara gärna sådant som kan antas vara oklart för läraren.

Du kan anta att det redan finns en funktion som heter **scan**, och att den returnerar typen på nästa token.

Uppgift 6: Scanning och reguljära uttryck (1 p)

Skriv reguljärt uttryck ("regexp") för tidsangivelserna i inmatningsspråket ovan. De består av timmar, minuter och sekunder, med kolon emellan. Sekunddelen kan innehålla decimaler.

Uppgift 7: Optimering (5 p)

Här är lite treadresskod:

```
(1) temp1 = b * 17
(2) temp2 = c * 18
(3) a = temp1 + temp2
(4) temp3 = c * 18
(5) if (a > temp3) goto 14
(6) temp4 = c - 1
(7) if (a > temp4) goto 13
(8) temp5 = 1 * d
(9) c = c - temp5
(10) temp6 = b * 17
(11) a = a + temp6
(12) goto 6
(13) goto 16
(14) e = b * 17
(15) f = c * 18
(16) temp7 = b * 17
(17) g = a + temp7
```

a) Dela in koden i basic blocks.

b) Optimera koden med hjälp av de vanliga metoderna för optimering, såväl inom ett enskilt basic block som i flera block. Beskriv inte bara resultatet, utan de olika stegen, gärna med namn.
